SPASE-QL Query Formation

Explanation

A SPASE-QL query is an XML document based upon the SPASE metadata model. Before creating a query you should be familiar with the basics of XML. (reference:http://www.w3schools.com/xml/default.asp). Each SPASE-QL query must be a valid XML document.

Description

The SPASE-QL query file follows a strict format containing descriptive and inquisitive information which is based upon the spaseql query schema located here: http://spaseql.gsfc.nasa.gov/schema/spaseql_query.xsd. The structure of the file is broken down into two main parts: context and request. The context being descriptive and the request being inquisitive or, in fact, the actual query.

Context

The context section is a data definition that contains historical information about the request. Such information includes time sent, where it is going and person responsible for sending the query. The context section is used solely for logging purposes. Its child elements are “RequestTime”, “Destination” and “Origin”.

Request

The request section contains information which will tell the server what data you are requesting. It is important to note that not all service implementations support all SPASE-QL features, and it is the responsibility of the service provider to make this known.

The request contains two main sections: select and where. The “Select” children elements will be the result(s) returned in the response file. Based upon the service's implementation a user can use the SelectOptions to populate this section of their query. This is done by enclosing each requested item in the element “SPASEQLTerm”. SelectOptions are found in the service's configuration file.

The “Where” element is the most complex and dynamic part of your query. This is how you will be able to fine tune your question to receive your custom result file. Each service uses a different subset of conditions, constraints and query formation. In a nut shell each service has the option to provide simple and complex queries, and may or may not implement all select options and constraints. This discrepancy spawns from the SPASE-QL api customization elasticity.

Simple Query

Starting with a simple query, you will be using the “Clause” element. The clause element itself has attributes and holds expression children for query customization. Attributes of the clause element include: ID, conditionsOccur, LogicalOperator, StartDate and StopDate. These attributes will be explained later in the document. Contained in the clause are one or more “Expression” elements. Expressions are the meat of your query. “Expression” elements are built using “Constraint” family of elements which have their own restrictions for query tuning.

Complex Query

If implemented, a Complex Query will be a combination of 1 or more “ComplexComponent” definitions. The “ComplexComponent” element is used as a container to hold “ComplexQuery” elements . Complex Components are made up of the “Clause” elements described above. A “ComplexComponent” has the same attributes as the “Clause” element. Instead of these attributes taking action on the expressions they will define how each inner clause is related to one another.

Expressions

Expressions are the portion of a SPASE-QL document that contain information about what you are querying for. They are enclosure elements that house the constraints. Each expression is governed by its parent element's attributes. For example StartDate and StopDate, ConditionsOccur or LogicalOperator attributes will define how the service handles each clause and its enclosed expressions.

Constraints

In this data definition a constraint will be the determination of what data you are requesting. There are many different constraints and each acts differently. Some constraints are complex; meaning they have their own children. For a detailed list of constraints please review the query schema. Constraints operate together with restrictions.

Constraint Restrictions

Restrictions allow you to fine tune your results even further. The current options of Constraint Restrictions are: “LessThan”, “GreaterThan”, “Equal”, and “Like”. The elements “LessThan” and “GreaterThan” have the attribute inclusive. Using the inclusive attribute will allow you to query for an equals to or simply a less than or greater than result.

Breaking down elements based upon the SPASE-QL Query Schema

Please refer to the SPASE-QL query schema documentation file for aid in creating queries and a breakdown of elements with their corresponding relationships There is an HTML representation of this file which can be found here: http://spaseql.gsfc.nasa.gov/schema/spaseql_query.xsd.html.

A bare bones query file contains the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<SPASEQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://spaseql.gsfc.nasa.gov/schema/spaseql_query.xsd"
 xmlns="http://www.spase-group.org/spaseql/Query">

   <SPASEQL_Version>1.0</SPASEQL_Version>
   <Context>
      <RequestTime></RequestTime>
      <Destination></Destination>
      <Origin></Origin>
   </Context>
   <Request>
      <Select>
         <SPASEQLTerm></SPASEQLTerm>
      </Select>
      <Where>
            <Clause LogicalOperator="and">
              <Expression></Expression>
            </Clause>
      </Where>
   </Request>
</SPASEQL>
```

Standard XML namespace and schema.

Figure 1: XML Declaration

The “Context” element.

Figure 2: Context Element

The “Request” element

Figure 3: Request Element

A basic population of the “Context” section would be as follows:

```
   <Context>
      <RequestTime>2009-02-03T12:00:00Z</RequestTime>
      <Destination>spase://VHO/api</Destination>
      <Origin>emailAddress | spase://Person/Your.Name</Origin>
   </Context>
```

The Request time should be a UTC time stamp of when the query was created/sent.
Destination will be a descriptive identifier of the destination.
Origin is a valid identifier of who you are.

A basic population of the “Select” section would be as follows:

```
      <Select>
         <SPASEQLTerm>GranuleID</SPASEQLTerm>
      </Select>
```

In the Select definition you can define one or more SPASEQLTerm's depending on the service's implementation.

A basic population of the “Where” section would be as follows:

```
<Where>
    <Clause ID=”Cadence”  LogicalOperator="and"
            StartDate=”1999-01-02T00:00:00Z”
            StopDate=”1999-03-01T00:00:00Z”
            ConditionsOccur=”atSameTimeInSameProduct”>
        <Expression>
            <Cadence><LessThan inclusive="yes">PT10S</LessThan></Cadence>
        </Expression>
        <Expression>
            <MeasurementType>MagneticField</MeasurementType>
        </Expression>
    </Clause>
</Where>
```

These Expressions use Constraints Cadence and MeasurementType. Cadence allows the restriction of LessThan, while MeasurementType is an enumeration field.

Figure 4: Expression Children

This clause implies that the query will return all elements that have a cadence of less than and equal to PT10S with a measurement type of magnetic field. If the LogicalOperator had been “or” your results would have been either a cadence less than and equal to PT10S OR a measurement type of magnetic field. There would be no need for the expressions to coincide which is dissimilar when used with the AND LogicalOperator.

The attribute “LogicalOperator” defines how the Expressions will interact and is required in each definition of “Clause”. As mentioned above a “Clause” element also has attributes “ID”, “conditionsOccur”, “StartDate” and “StopDate”. These attributes are optional, but describe or affect the outcome of the query.

The “ID” attribute is a descriptive term that can be used as a locater for each “Clause” element. Generally this is useful for the service provider, but is also helpful for you to define sections in your query. “conditionsOccur” defines how the logicalOperator will function. In space physics context “and” is ambiguous. There are two options for conditionsOccur, “atSameTime” and “atSameTimeInSameProduct”. Essentially this means do “A” and “B” occur at just the same time or do “A” and “B” occur at the same time and in the same product; where “A” and “B” are expressions. The default is “atSameTimeInSameProduct”. The optional “StartDate” and “StopDate” govern the time interval for which the queried data will be gathered from.

Notice that the expression statements include constraints and constraint restrictions. Fine tuning your constraints with constraint restrictions is often necessary depending on which type of constraint you are querying on. See Figure 4 “Expression Children”.

Creating a Complex Query

```
<Where>
    <ComplexClause>
      <ComplexComponent LogicalOperator=”and”>
        <Clause ID=”Cadence” LogicalOperator="and">
          <Expression>
            <Cadence><LessThan inclusive="yes">PT10S</LessThan></Cadence>
          </Expression>
          <Expression>
            <MeasurementType>MagneticField</MeasurementType>
          </Expression>
        </Clause>
        <Clause ID="Coordinates" LogicalOperator="and">
          <Expression>
            <StatisticalEstimator>
              <Component>Vx</Component>
              <CoordinateSystemName>GSM</CoordinateSystemName>
              <Units>km/s</Units>
              <Estimator>Minimum</Estimator>
              <LessThan inclusive="no">-200</LessThan>
            </StatisticalEstimator>
          </Expression>
          <Expression>
            <StatisticalEstimator>
              <Component>Vx</Component>
              <CoordinateSystemName>GSM</CoordinateSystemName>
              <Units>km/s</Units>
              <Estimator>Maximum</Estimator>
              <GreaterThan inclusive="no">200</GreaterThan>
            </StatisticalEstimator>
          </Expression>
        </Clause>
      </ComplexComponent>
    </ComplexClause>
</Where>
```

Note this expression uses the constraint StatisticalEstimator. StatisticalEstimator is a complex constraint. Please see “Breaking down the Constraint Statistical Estimator”

Figure 5: Statistical Estimator

A complex clause returns results based upon its inner clause conditions. It is based upon the top level ComplexComponent attribute LogicalOperator. In this case we have “and” as the LogicalOperator. This implies that all clauses must be met in order for results to be returned.

Breaking down the Constraint Statistical Estimator

The “StatisticalEstimator” Constraint is a multi-faceted element with several children. You will see other Constraints which also have children similar to this one. The “Component” element would be your axises. Where the “CoordinateSystem” is self explanatory, but is the coordinate system of the space craft with respect to its orbit. In this case the element “Units” is used as a velocity determination, but can also be used to reference other types of unit measurements when used with other constraints. The “Estimator” element type is an enumeration restriction that determines the numerical data queried. This usage specifies the minimum. For reference, at present the complete list of estimator options are “Average'|'Median'|'Minimum'|'Maximum'|'StandardDeviation'|'DataAvailability”. Finally the “LessThan” constraint restriction with the attribute “inclusive” and value of “no” means that the results will show anything meeting all conditions of the expression which are also less than and not equal to -200.

Completing the Complex Query

Note how to encapsulate the clause when conducting a complex query.

```
<ComplexClause>
    <ComplexQuery>
        <Clause />
        <Clause />
        <Clause />
    </ComplexQuery>
</ComplexClause>
```

Tip when using Complex Queries

If you decide to use a complex query, you can often place a simple query above or below your complex query definition. This allows you the to get extra result sets in your result file.

Conclusion

In the end it boils down to getting familiar with the SPASE-QL schema file and the structure of the SPASE-QL query file. It is very important to have the schema on hand to aid with query formation. The document contains each possible data element along with a description of its presence in the SPASE Query Language. As a reference the SPASE-QL web site offers multiple query examples to get you started. Just visit each service's page for an example listing. It is highly encouraged to review and try all examples provided. You will be creating your own queries and getting the data you need in no time.

Using SPASE-QL will allow you to get the data you need from multiple services in a stream line, simple and robust fashion.

References

SPASE-QL website: http://spaseql.gsfc.nasa.gov

SPASE-QL services: http://spaseql.gsfc.nasa.gov/services/

SPASE-QL query schema: http://spaseql.gsfc.nasa.gov/schema/spaseql_query.xsd

SPASE-QL query schema (HTML format): http://spaseql.gsfc.nasa.gov/schema/spaseql_query.xsd.html